Towards the integration of security patterns in UML Component-based Applications

Anas Motii[1], Brahim Hamid[2], Agnès Lanusse[1], Jean-Michel Bruel[2]

[1] CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems, P.C. 174, Gif-sur-Yvette, 91191, France
{anas.motii,agnes.lanusse}@cea.fr
[2] IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
{brahim.hamid,bruel}@irit.fr

Abstract. In software engineering, design patterns are considered effective tools for the reuse of specific information. They are widely used today to provide architects and designers with reusable design knowledge. This paper is about the use of patterns in secure systems and software engineering, in particular in model based engineering. In this paper, we are proposing a model-based methodology for security pattern integration aimed to secure component-based software system architectures in UML. The methodology is based on merging techniques and verifications against integration constraints described in OCL language. The paper illustrates the method through a VPN security pattern.

Keywords: model-based methodology, model driven engineering, UML, OCL, security patterns, pattern integration, secure architecture

1 Introduction

On the last decade, distributed systems have grown in complexity and connectivity. This has led to the introduction of several security breaches such as the one in [1]. This is due to the fact that security has been treated at the late phases of development. The security community all agree that there is a need of secure software and systems methods that tackle security at early phases. Indeed, patterns represent generic solutions for recurring problems. The use of a well-documented catalogue of patterns has proven to be useful in several project in particular for security. However these catalogues are not enough, there is a huge need for methods for helping users select relevant patterns and apply them. Classical pattern concepts described in the pattern templates such as Gang Of Four (GoF) [2] are commonly accepted but we believe that the pattern conceptual model must embody additional concepts relevant for pattern selection and integration into architectures. In this paper, we propose a method for integrating security patterns within architectures. For that purpose, Model Driven Engineering (MDE) offers an ideal development context using metamodeling and model transformation. In fact, if we consider the pattern and the application as two models, the integration process can be automated as a MDE composition operation resulting in a new (refined) architecture. In this paper we focus on architectures described as UML composite structures. In [3], the authors have presented an automated process for selecting patterns based on a relational view of the application and pattern models. Their approach is similar to ours since it uses role assignment as a query to select software patterns.

2 Background: Prerequisites for using our process

The modeling language and the integration process used for security patterns are based on previous works related to [4]. Here, we present the pattern concepts we have defined to support our methodology.

Pattern: is the main concept. It encapsulates a *solution* of a recurring *problem* in specific *context*.

Integration relevant concepts: (from [4]**)**

- Pattern entity (role): The solution of the pattern consists of a set of entities (roles). Each entity of the targeted application will play a role in the pattern.
- Application: architecture targeted by the integration. It consists of a set of application entities. Not all entities must be a target for integration but only a subset. Some entities will play a role in the pattern.
- Plays (is bound to): is a concept used to link pattern roles and the domain application entities.
- Properties: represent the intent of the pattern and formulate the solution of the problem.
- Constraints: are assumptions that the application must satisfy in order for the pattern to deliver its properties. There are two kinds of constraints: preconditions and post conditions.

Traceability relevant concepts (added):

- Create: If some roles of the pattern are not played by the application, new elements are created. This concept is used to link the newly created elements and the corresponding pattern roles.
- Integration trace (pattern application): represents the success of the integration on one hand. On the other hand, it guarantees traceability as it keeps the “Plays” and the “Create” traces.

3 Pattern integration process

The process follows four phases: “Preparation” extracts from the pattern the solution description and its related constraints (preconditions and post conditions) into a UML model. Pattern properties and constraints are formalized using OCL[1]. “Elicitation” builds a bridge between the application and the pattern. “Merge” does the merging be-

[1] The Object Constraint Language adds constraint rules within UML models

tween them and generates an integration trace. Finally, “Adaptation” offers the possibility to make changes by letting the user refining the new application. We illustrate the approach with the application of a VPN pattern on a UML composite structure describing a distributed system. The scenario is performed as follows. A designer wants to establish a secure communication between two clients: “host1” and “host2” and a corporate network server “companyServer”. The designer starts by modeling the clients and corporate server in the left side of **Fig. 1**. Then he follows the steps described below to obtain a refined model integrating the VPN pattern.

Preparation.

Fig. 2 shows the VPN architecture solution as a UML composite structure along with its constraints. The VPN has the following roles: “client”, “VPNClientSide”, “VPN-NetworkSide”, “communicationNetwork” and “corporateServer”. As seen in **Fig. 2**, the constraints over the application are formalized in OCL. Each constraint is attached to the constrained elements. These constraints must normally be expressed by security experts. In this example they are mainly illustrative.

- Preconditions: (1) there must not be a connection between elements playing roles “client” and “corporateServer” expressed in the constraint “NoUnsecureCommunication”; (2) elements of the application cannot play the exclusive roles: “client” and “VPNNetworkSide”, “client” and “VPNClientSide”, and “communicationNetwork” and “corporateServer” (the last constraint is not shown in the figure).
- Post conditions (not shown in the figure): The application must not have a direct connection between entities playing roles “client” and “corporateServer” (no direct link). If there is an indirect link, it must go through the “VPNClientSide” first and then “VPNNetworkSide”.

Elicitation.

Role assignment (casting diagram). The designer specifies that “host1” and “host2” play the roles of “client”, and that “companyServer” plays the role “corporateServer”. Thus he draws the casting diagram in the right side of **Fig. 1**.
Role assignment validation. Once the casting has been made the designer checks its validity through the checking module against preconditions. If one of the preconditions is violated the casting is rejected. For example, if the designer had connected “host1” and “companyServer” directly, then “NoUnsecureCommunication” will be violated.

Merge.

Once the casting is correct, this phase consists of merging the composite structure of the pattern and the application, using merge points defined in the casting diagram, in order to obtain a new application as shown in the bottom of **Fig. 3**.
In addition, a trace is generated linking the new application model, and the previous application and pattern models as seen in **Fig. 3**. It consists of the castings (dependencies stereotyped with “Play”) in the “casting diagram” and dependencies toward the

newly created entities (dependencies stereotyped with “Create”). The trace is necessary for the last phase.

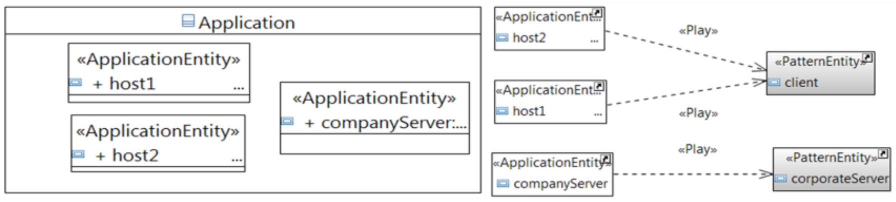
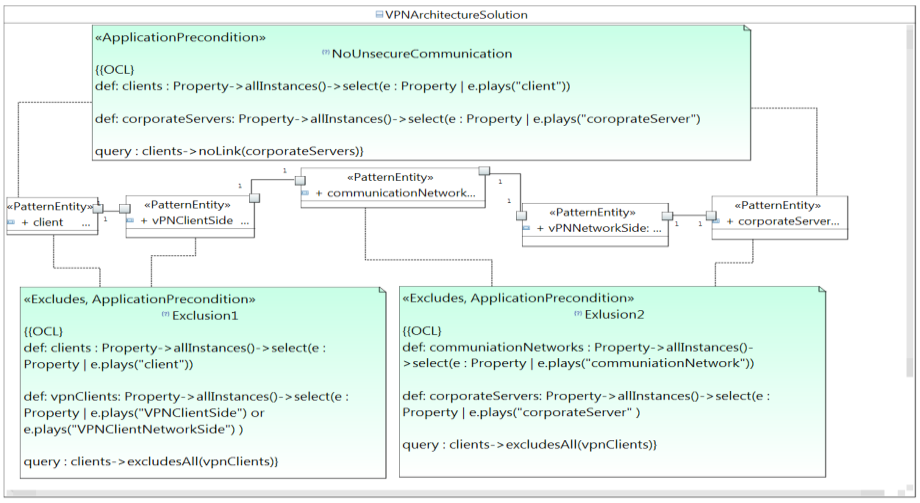

Application
«ApplicationEntity» + host1
«ApplicationEntity» + host2
«ApplicationEntity» + companyServer:...
«ApplicationEnt... host2
«ApplicationEnt... host1
«ApplicationEntity» companyServer
«Play»
«Play»
«Play»
«PatternEntity» client
«PatternEntity» corporateServer

Fig. 1. Application model (left side) and casting diagram (right side)


VPNArchitectureSolution
«PatternEntity» + client
«PatternEntity» + vPNClientSide
«PatternEntity» + communicationNetwork...
«PatternEntity» + vPNNetworkSide: ...
«PatternEntity» + corporateServer...
1
1
1
1
1
1
1
1

«ApplicationPrecondition»
NoUnsecureCommunication

```
{{OCL}
def: clients : Property->allInstances()->select(e : Property | e.plays("client"))

def: corporateServers: Property->allInstances()->select(e : Property | e.plays("coroprateServer")

query : clients->noLink(corporateServers)}
```

«Excludes, ApplicationPrecondition»
Exclusion1

```
{{OCL}
def: clients : Property->allInstances()->select(e :
Property | e.plays("client"))

def: vpnClients: Property->allInstances()->select(e :
Property | e.plays("VPNClientSide") or
e.plays("VPNClientNetworkSide") )

query : clients->excludesAll(vpnClients)}
```

«Excludes, ApplicationPrecondition»
Exlusion2

```
{{OCL}
def: communiationNetworks : Property->allInstances()-
>select(e : Property | e.plays("communiationNetwork"))

def: corporateServers: Property->allInstances()->select(e :
Property | e.plays("corporateServer" )

query : clients->excludesAll(vpnClients)}
```

Fig. 2. VPN solution architecture model

Adaptation.

The adaptation phase consists in checking the resulting application model against the pattern post conditions at each model change. This change can result from ad-hoc tailoring by the designer or from the integration of another security pattern (e.g., firewall pattern). For this reason, the created trace plays an important role in this phase. Since the trace keeps a link between the refined application, and the previous version of the application and the pattern. The verification of post conditions over the new application is guaranteed by the checking module. Indeed, at each change in the new application, the checking module verifies that the post conditions described in the pattern are still true in the elements playing a role or created after the integration. For example, if applying a new pattern over the new application creates a connection between “host1” and “companyServer”, then the post condition “no direct link” will be violated. It should be noted that at this phase all the post conditions of patterns that have already been applied are rechecked to enforce their continuous functionality.

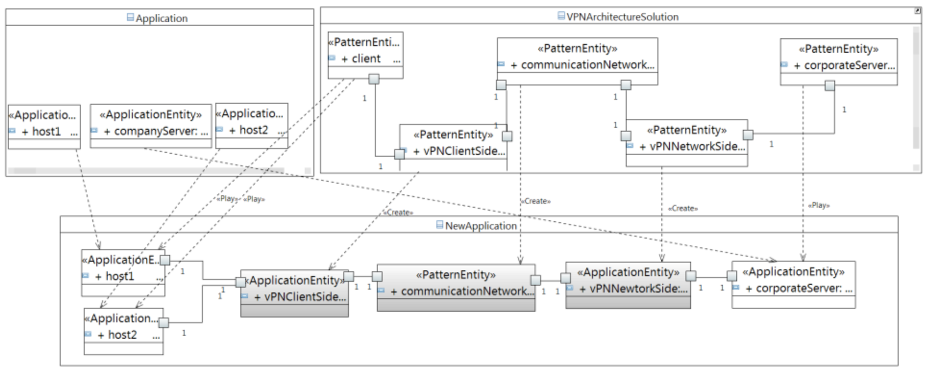


Fig. 3. Resulting application structure with traceability links

4 Conclusion

This paper presents a pattern integration process, aimed at correctly applying security patterns on systems and software architectures. The process has been illustrated through the application of a VPN pattern. In future work, we will apply the process through several security patterns.

The challenges faced during the integration process were due to the fact it is part of global MDE process for securing system architectures. It is composed of several phases: (1) risk analysis, (2) pattern selection with regards to multi-concern objectives, (3) pattern integration. In order to guarantee fast considerations of changes in requirements such as: real-time, we thought of adding new concepts such as traceability links for the architect to keep a trace of the added components and the applied pattern. For instance, if the real-time requirements change, some selected patterns may change. Thanks to traceability, the replacement of patterns becomes easier since the architect has detailed information of the currently applied pattern.

Acknowledgements. This work is conducted in the context of a Ph.D. thesis funded by CEA LIST and co-leaded by CEA (LISE) and IRIT (MACAO).

Reference

1. Attackers Alter Water Treatment Systems in Utility Hack: Report | SecurityWeek.Com, http://www.securityweek.com/attackers-alter-water-treatment-systems-utility-hack-report.
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995).
3. van den Berghe, A., Van Haaren, J., Van Baelen, S., Berbers, Y., Joosen, W.: Towards an automated pattern selection procedure in software models. In: Late breaking papers of the 22nd international conference on inductive logic programming (ILP 2012). pp. 68–73. CEUR-WS (2013).
4. Hamid, B., Percebois, C., Gouteux, D.: A Methodology for Integration of Patterns with Validation Purpose. In: Proceedings of the 17th European Conference on Pattern Languages of Programs. p. 8:1–8:14. ACM, New York, NY, USA (2012).